

(19) World Intellectual Property
Organization
International Bureau



(43) International Publication Date
1 December 2005 (01.12.2005)

PCT

(10) International Publication Number
WO 2005/114238 A1

(51) International Patent Classification⁷: **G01R 31/319**

[US/US]; 1777 Merlot Way, Salinas, California, 93906 (US). **PRAMANICK, Ankan** [US/US]; 5839 Capilano Drive, San Jose, California, 95318 (US).

(21) International Application Number:
PCT/JP2005/009810

(74) Agent: **RYUKA, Akihiro**; 6F, Toshin Building, 24-12, Shinjuku 1-chome, Shinjuku-ku, Tokyo 1600022 (JP).

(22) International Filing Date: 23 May 2005 (23.05.2005)

(25) Filing Language: English

(81) Designated States (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(26) Publication Language: English

(30) Priority Data:
60/573,577 22 May 2004 (22.05.2004) US
10/917,821 13 August 2004 (13.08.2004) US

(71) Applicant (*for all designated States except US*): **ADVANTEST CORPORATION** [JP/JP]; 1-32-1, Asahi-cho, Nerima-ku, Tokyo, 1790071 (JP).

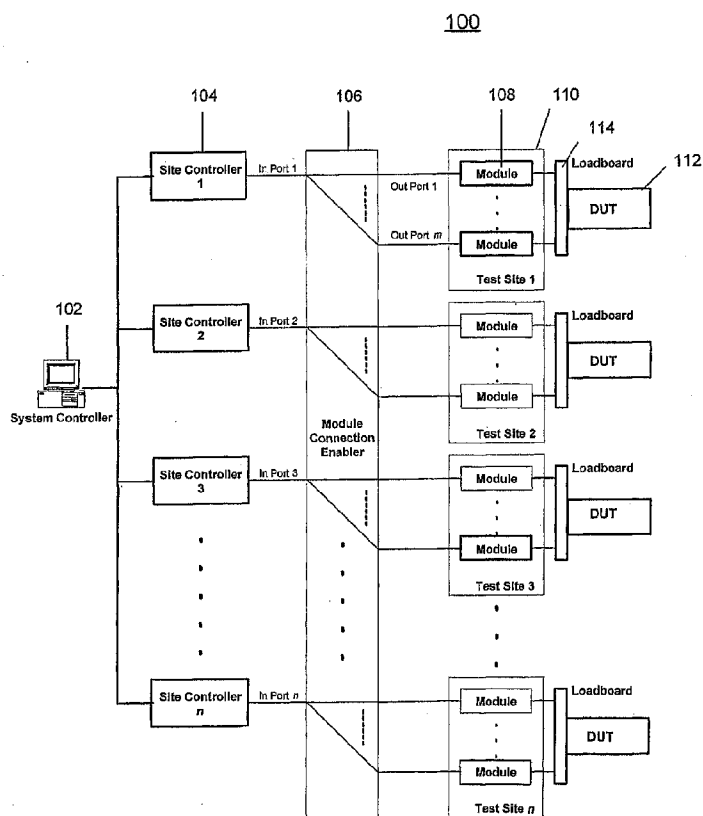
(84) Designated States (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GI, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM,

(72) Inventors; and

(75) Inventors/Applicants (*for US only*): **ELSTON, Mark**

[Continued on next page]

(54) Title: **DATALOG SUPPORT IN A MODULAR TEST SYSTEM**



(57) Abstract: A method for communicating test information from a source to a destination is disclosed. The method includes providing a modular test system, where the modular test system comprises a system controller for controlling at least one site controller, the at least one site controller for controlling at least one test module. The method further includes providing a datalog framework for supporting extension of user-defined datalog formats, providing support classes for supporting user-initiated datalog events, receiving a datalog event requesting for communicating input test information from the source to the destination, configuring output test information based upon the destination, the datalog framework and the support classes, and transferring the output test information to the destination.



ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *with international search report*

DESCRIPTION

DATALOG SUPPORT IN A MODULAR TEST SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

- 5 [0001] This application claims the benefit of provisional application no. 60/573,577, "Software Development in an Open Architecture Test System," filed by Advantest Corporation on May 22, 2004, which is incorporated herein in its entirety by reference.

FIELD OF THE INVENTION

- [0002] The present invention relates to the field of automated test equipment (ATE).
10 In particular, the present invention relates to a method and system for supporting datalog in an open architecture test system.

BACKGROUND OF THE INVENTION

- [0003] The increasing complexity of System-on-a-Chip (SOC) devices and the simultaneous demand for a reduction in the cost of chip testing has forced both integrated
15 circuit (IC) manufacturers and tester vendors to rethink how IC testing should be performed. According to industry studies, without re-engineering the projected cost of testers will continue to rise dramatically in the near future.

- [0004] A major reason for the high cost of test equipment is the specialized nature of conventional tester architecture. Each tester manufacturer has a number of tester platforms
20 that are not only incompatible across companies such as Advantest, Teradyne and Agilent, but also incompatible across platforms within a company, such as the T3300, T5500 and T6600 series testers manufactured by Advantest. Because of these incompatibilities, each tester requires its own specialized hardware and software components, and these specialized hardware and software components cannot be used on other testers. In addition,
25 a significant effort is required to port a test program from one tester to another, and to develop third party solutions. Even when a third party solution is developed for a platform, it cannot be ported or reused on a different platform. The translation process from one platform to another is generally complex and error prone, resulting in additional effort, time and increased test cost.

[0005] Datalogging is used to provide status information to the user when running a test or a series of tests. The information reported in a datalog may include the pass/fail status of the device under test (DUT), any relevant measured parameters, and the overall run status of the test itself. This information is typically used offline to assess the completion of a test run and the performance of the device being tested. The support of datalog capabilities allows users to output their test information from a designated source to a designated destination in a desired format.

[0006] One of the problems of the specialized tester architecture is that all hardware and software remain in a fixed configuration for a given tester. To test a hardware device or an IC, a dedicated test program is developed that uses some or all of the tester capabilities to define the test data, signals, waveforms, and current and voltage levels, as well as to collect the DUT response and to determine DUT pass/fail.

[0007] Since a test system needs to exercise a wide range of functionalities and operations in order to test a wide variety of test modules and their corresponding DUTs, there is a need for an open architecture test system that can be configured to support the wide variety of test modules. Specifically, in order to support the wide variety of test modules, there is a need for a datalog framework within the open architecture test system that can be configured to work with the different formats of the different sources and destinations of the test system.

SUMMARY

[0008] In one embodiment of the present invention, a method for communicating test information from a source to a destination includes providing a modular test system, where the modular test system comprises a system controller for controlling at least one site controller, the at least one site controller for controlling at least one test module. The method further includes providing a datalog framework for supporting extension of user-defined datalog formats, providing support classes for supporting user-initiated datalog events, receiving a datalog event requesting for communicating input test information from the source to the destination, configuring output test information based upon the destination, the datalog framework and the support classes, and transferring the output test information to the destination.

[0009] In another embodiment of the present invention, a modular test system includes a system controller, at least one site controller coupled to the system controller, at least one

test module and its corresponding device under test (DUT), a datalog framework configured to support extension of user-defined datalog formats, and one or more support classes configured to support user-initiated datalog events. The modular test system further includes means for receiving a datalog event requesting for communicating input
5 test information from the source to the destination, means for configuring output test information based upon the destination, the datalog framework and the support classes, and means for transferring the output test information to the destination.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The aforementioned features and advantages of the invention as well as
10 additional features and advantages thereof will be more clearly understood hereinafter as a result of a detailed description of embodiments of the invention when taken in conjunction with the following drawings.

[0011] Figure 1 illustrates an open architecture test system according to an embodiment of the present invention.

15 [0012] Figure 2 illustrates an implementation of a datalog framework according to an embodiment of the present invention.

[0013] Figure 3 illustrates an implementation of datalog sources according to an embodiment of the present invention.

20 [0014] Figure 4 illustrates a method for registering source types according to an embodiment of the present invention.

[0015] Figure 5 illustrates a method for formatting and streaming data according to an embodiment of the present invention.

[0016] Figure 6 illustrates an implementation of enabling datalogging dynamically according to an embodiment of the present invention.

25 [0017] Figure 7 illustrates an implementation of disabling datalogging dynamically according to an embodiment of the present invention.

[0018] Figure 8 illustrates an implementation of modifying datalog format dynamically according to an embodiment of the present invention.

30 [0019] Figure 9 illustrates an implementation of assigning datalog output streams dynamically according to an embodiment of the present invention.

[0020] Figure 10 illustrates an implementation of disabling datalog output stream dynamically according to an embodiment of the present invention.

[0021] Figure 11 illustrates an implementation of adapting new source dynamically according to an embodiment of the present invention.

DESCRIPTION OF EMBODIMENTS

[0022] Methods and systems are provided for datalog support in a modular test system.

5 The following description is presented to enable any person skilled in the art to make and use the invention. Descriptions of specific techniques and applications are provided only as examples. Various modifications to the examples described herein will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other examples and applications without departing from the spirit and scope of
10 the invention. Thus, the present invention is not intended to be limited to the examples described and shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0023] Figure 1 illustrates an open architecture test system according to an embodiment of the present invention. A system controller (SysC) 102 is coupled to
15 multiple site controllers (SiteCs) 104. The system controller may also be coupled to a network to access associated files. Through a module connection enabler 106, each site controller is coupled to control one or more test modules 108 located at a test site 110. The module connection enabler 106 allows reconfiguration of connected hardware modules 108 and also serves as a bus for data transfer (for loading pattern data, gathering response data,
20 providing control, etc.). In addition, through the module connection enabler, a module at one site can access a module at another site. The module connection enabler 106 allows different test sites to have the same or different module configurations. In other words, each test site may employ different numbers and types of modules. Possible hardware implementations include dedicated connections, switch connections, bus connections, ring
25 connections, and star connections. The module connection enabler 106 may be implemented by a switch matrix, for example. Each test site 110 is associated with a DUT 112, which is connected to the modules of the corresponding site through a loadboard 114. In another embodiment, a single site controller may be connected to multiple DUT sites.

[0024] The system controller 102 serves as the overall system manager. It coordinates
30 the site controller activities, manages system-level parallel test strategies, and additionally provides for handler/probe controls as well as system-level data-logging and error handling support. The system controller 102 is the primary point of interaction for a test engineer in

verifying and debugging the test environment. It provides a gateway to the site controllers 104, and manages the synchronization of the site controller activities in a multi-DUT environment. It further runs user applications and tools, such as the Datalog graphical user interface (DatalogGUI). Depending on the operational setting, the system controller 102 can be deployed on a CPU that is separate from the operation of site controllers 104. Alternatively a common CPU may be shared by the system controller 102 and the site controllers 104. Similarly, each site controller 104 can be deployed on its own dedicated CPU (central processing unit), or as a separate process or thread within the same CPU.

[0025] The site controllers 104 are responsible for running a test plan to test the DUTs.

10 The test plan creates specific tests by using the Framework Classes as well as standard or user supplied Test Classes that encapsulate the test methodology. In addition, the test plan configures the hardware using the Standard Interfaces, and defines the test flow.

[0026] The system architecture of the present invention may be conceptually envisioned as the distributed system shown in Figure 1 with the understanding that the individual system components may also be regarded as logical components of an integrated, monolithic system, and not necessarily as physical components of a distributed system. The plug-and-play or replaceable modules are facilitated by use of standard interfaces at both hardware and software levels. A tester operating system (TOS) allows a user to write test plan programs using a test plan programming language, and to operate the test system in a way specific to a particular device under test (DUT). It also allows the user to package sequences of the test system operations commonly used in test plan programs as libraries. These libraries are sometimes referred to as test classes and test templates.

Datalog Framework

[0027] The datalog framework runs on individual Site Controllers, along with the test plan and the system framework classes. It provides a set of interfaces, classes and methods to support the development, deployment and management of the datalog system. The datalog framework

- manages datalog elements, such as sources, streams and formatters;
- dispatches datalog events;
- 30 • provides an application programming interface (API) that allows the creation of new datalog events;
- provides APIs to set up and control datalogging;

- provides a general purpose formatter to handle common formatting needs; and
- provides support for the integration of modular third party and/or customer datalog services, which makes the datalog framework open.

5 Standard Datalog Interfaces

[0028] Figure 2 illustrates an implementation of a datalog framework according to an embodiment of the present invention. The set of relationships among the datalog framework interfaces and support classes are illustrated through the unified modeling language (UML) class diagrams of Figure 2. The datalog framework 200 includes a datalog stream interface 202, a datalog format interface 204, a datalog type interface 206, and a datalog source interface 208. The datalog framework 200 further includes a datalog manager class 210, a datalog format map class 212, a datalog format group class 214, a datalog values class 216, a general format class 218, and a user format class 220.

[0029] The standard datalog interfaces to the tester operating system (TOS) are defined as pure abstract C++ classes. The datalog stream interface (IDatalogStream) 202 represents an output file or device. The datalog system sends formatted output to streams. The system provides two built-in streams, a FileStream for sending output to a local disk file, and a ConsoleStream for sending output to the System's Console application.

[0030] The datalog format interface (IDatalogFormat) 204 represents the necessary instructions required to format data associated with datalog events to an output. The specific nature of the formatting capability is hidden inside each implementation of the IDatalogFormat. For example, the built-in GeneralFormat allows a general format string to be used with a macro-like capability for extracting named values from the DatalogValues objects, and inserting them into a formatted string.

[0031] The datalog type interface (IDatalogType) 206 represents a class or a "type" of datalog source, such as a particular type of a Test. The DatalogManager 210 maps events to formats and streams based in part on the type of the source of the event. As new types of sources are created, they register themselves with the DatalogManager 210 and receive an identifier (ID) value. Managing these types allows the DatalogManager 210 to enable/disable whole classes of datalog sources as well as specific instances of datalog sources. The IDatalogType objects 206 may generate multiple classes of datalog events. For example, a user test that loops over a set of patterns until some condition occurs may

generate TestStart, TestEnd, IterationStart, IterationEnd, and ConditionTest datalog events. Each of these classes of events may have different data fields associated with it (see DatalogValues class below).

[0032] The datalog source interface (IDatalogSource) 208 represents a source of datalog events. While the IDatalogType 206 characterizes an entire class of datalog source objects, the IDatalogSource 208 represents an individual source instance. This interface allows enabling/disabling datalogging of individual sources.

Datalog Support Classes

[0033] A central object of the datalog framework 200 is the DatalogManager object 210. This object is responsible for maintaining all datalog output streams as well as formatting and forwarding datalog events and their associated data to the appropriate stream(s). The DatalogManager 210 resides in the test plan server (TPS) of the Site Controller. It is the datalog execution engine. Specifically, the main functions of the DatalogManager 210 include:

- 15 • Managing a set of global datalog streams (i.e., objects implementing the IDatalogStream interface).
- Managing a set of datalog formats (i.e., objects implementing the IDatalogFormat interface), each of which generates a formatted string as a result of a set of datalog events.
- 20 • Distributing the datalog events to all associated datalog streams. All destinations for streams can be controlled through the DatalogManager.
- Maintaining and controlling Datalog Mask conditions, which have the effect of filtering datalog output. Note that the application of the mask may occur at the source to improve system performance.
- 25 • Starting up/cleaning up the datalog system.
- Enabling/disabling the entire datalog system, as well as individual datalog types or sources.

[0034] The datalog format map class 212 is a named <EventType, EventName, Format> combination. That is, it groups together an EventType (representing a specific IDatalogType object), an EventName (representing one of the datalog events generated by EventType objects) and a Format (representing an IDatalogFormat object).

[0035] The datalog format group class 214 is a collection of DatalogFormatMap objects. Users may bundle DatalogFormatMap objects into a single group and assign the group to a stream or several streams. This provides the user-level control over the ultimate formatting of datalog events and the routing of these formatted strings to the appropriate streams.

[0036] The datalog values class 216 contains header information, like datalog type identifier and event, and a list of datalog fields that are set by the event source. The DatalogValues object acts as a string-to-string map, from name to value, allowing format objects to extract the required values by name, and insert them into a formatted stream.

10 [0037] The general format class 218 is an implementation of IDatalogFormat. It allows a general format string to be used with a macro-like capability for extracting named values from the DatalogValues objects, and inserting them into a formatted string.

[0038] The user format class 220 is a placeholder for a user-defined format class. If the supplied GeneralFormat class does not meet a user's specific requirements, one may
15 add a new format class to the system and make use of it.

Datalog Sources

[0039] In one embodiment, a Site Controller based object may act as a source of datalog events by implementing an IDatalogSource interface 208. For example, the Test Plan and Test objects are examples of sources of events. Each instance of such an object
20 implements the IDatalogSource interface. In addition, each type of such an object, for example FunctionalTest, has an implementation of the IDatalogType interface 206 associated with it. This is accomplished with a class-wide static object for each type of datalog source. Figure 3 illustrates an implementation of datalog sources according to an embodiment of the present invention. The datalog sources include a functional test () class
25 302, a datalog type implementation () class 304, the datalog type interface 306 and the datalog source interface 308.

[0040] As shown in Figure 3, the FunctionalTest class 302 has a class-wide static member of type DatalogTypeImpl 304. An instance of the FunctionalTest checks this member variable when it is created. If the variable has not been created and assigned, then
30 the constructor creates an instance of DatalogTypeImpl, initializes it, assigns it to the static member variable, and registers it with the DatalogManager. A subsequent instance of the FunctionalTest class accesses the DatalogTypeImpl data member, and adds itself (i.e., the

instance of the FunctionalTest as an IDatalogSource) to DatalogTypeImpl's list of datalog source instances that the latter FunctionalTest instance is serving.

[0041] In another embodiment, a sub-class of the FunctionalTest class, for example ADifferentFunctionalTest, may attempt to establish a datalog type different from that of the FunctionalTest. An implementation of ADifferentFunctionalTest may accomplish this by not using the class-wide DatalogTypeImpl associated with the FunctionalTest, but using its own class-wide instance of a DatalogTypeImpl. Note that each instance of FunctionalTest implements the IDatalogSource interface. DatalogTypeImpl uses this interface to enable/disable individual sources.

10 Source Type Registration

[0042] When a Test Plan is loaded, datalog event sources may register with the DatalogManager object. The return value from this registration is an identifier, which is then used to generate events.

[0043] Figure 4 illustrates a method for registering source types according to an embodiment of the present invention. The group of collaborating objects for registering source types includes a Test Plan object 402, a FunctionalTest1 object 404, a FunctionalTest2 object 406, and a DatalogManager object 408. Note that when the Test Plan creates two test instances of the same type (FunctionalTest1 and FunctionalTest2), the first test instance registers a new datalog type with the DatalogManager.

[0044] Users may set datalog formats for different event types. This may be done during Test Plan initialization to provide default formats and/or done interactively from remote applications. A similar procedure is followed to allow users to set the datalog streams. The DatalogManager maintains a mapping from a set of {datalog-type, datalog-event} to a corresponding set of {datalog-format, datalog-stream} that specifies the route an event of a particular type may take through the test system. For file-based streams, a Test class on the Site Controller may close the stream allowing the datalog file to be transferred to the System Controller. The stream may then be reopened or replaced with another stream.

30 Formatting and Streaming Data

[0045] During testing, an object may generate a datalog event that is logged to a data stream. Figure 5 illustrates a method for formatting and streaming data according to an

embodiment of the present invention. The group of collaborating objects and interfaces for performing this task include a Test Plan object 502, a DatalogValues object 504, a DatalogManager object 506, a datalog format interface (IDatalogFormat) 508 and a datalog stream interface (IDatalogStream) 510. The Test Plan 502 creates the DatalogValues object instance 504, sets the appropriate values by calling the setValue() method, and passes this object to the doDatalog() method of the DatalogManager 506. The doDatalog() method then finds the appropriate datalog format object 508 for the type/event combination, and calls the apply() method on this datalog format object 508 to obtain a string object in return. This string object is then passed to the associated datalog stream object 510 for output using the writeMessage() method.

Datalog Initialization

[0046] In a different embodiment, the datalog system is initialized using the following steps:

- create datalog stream(s) and add them to the DatalogManager.
- create datalog format(s) and add them to the DatalogManager.
- create datalog format map(s) and add them to the DatalogManager.
- create datalog format group(s) and add them to the DatalogManager.

[0047] The datalog application programming interface (API) provides functions to perform the above tasks. In addition, the test system provides a Test class, DatalogSetupTest, which reads one or more configuration files and performs the above steps. The example below illustrates the use of the DatalogSetupTest configuration file to perform the initialization steps. The DLL parameter within the datalog stream, datalog format map and datalog format group specifies a library to use for implementing the IDatalogStream 202 or IDatalogFormat 204.

```

25      Version 0.1.0;
      # Enable the datalog system
      Enabled;

      # Step 1: Stream Creation

30      # Creates a stream of type FileStream, which logs the message to
      # a file.
      Stream FunctionalTestStream
      {
35          DLL "FileStream";
          FileName "FuncDatalogDut<DutID>.log";

```

```

        Overwrite "1";
    }

    # Step 2: Format Creation

5    # Adds a format with a line that looks like:
    #   Signal Info: $SignalInfo
    # where $SignalInfo is replaced by the string provided in the datalog
    # source.
10    Format SignalInfoFormat
    {
        DLL "GeneralFormat";
        Format "Signal Info: $SignalInfo";
    } # Adds a format with a line that looks like:
15    #   Test Result: $Result
    # where $Result is replaced by the string provided in the datalog
    # source.
    Format TestResultFormat
    {
20        DLL "GeneralFormat";
        Format "Test Result: $Result";
    }

    # Step 3: FormatMap Creation

25    # Maps format SignalInfoFormat to event DumpSignal from sources
    # of type com.Advantest.oai.TestClasses.FunctionalTest.
    FormatMap SignalFormatMap
    {
30        Type com.advantest.oai.TestClasses.FunctionalTest;
        Event DumpSignal;
        Format SignalInfoFormat;
    }

35    # Maps format TestResultFormat to event TestResult from sources
    # of type com.Advantest.oai.TestClasses.FunctionalTest.
    FormatMap ResultFormatMap
    {
40        Type com.advantest.oai.TestClasses.FunctionalTest;
        Event TestResult;
        Format TestResultFormat;
    }

    # Step 4: FormatGroup Creation

45    # Creates a group which directs messages from format map
    # SignalFormatMap and ResultFormatMap to destination FunctionalTestStream
    # and ConsoleStream.
    FormatGroup FunctionalTestGroup
    {

```

```

    FormatMap SignalFormatMap;
    FormatMap ResultFormatMap;

    Stream FunctionalTestStream;
5    Stream ConsoleStream;
    }

```

Semantics of Datalog Setup File

[0048] In yet another embodiment, a datalog setup file includes four different types of blocks: datalog format blocks, datalog stream blocks, datalog format map blocks and
 10 datalog format group blocks. These blocks are used to define the datalog formats, datalog streams, datalog format maps and datalog format groups respectively. Any of these four blocks may be used multiple times in a same setup file, and the datalog system may also be set up with multiple separate datalog setup files. Thus, the datalog system setup is modular and flexible. For example, there may be a first datalog setup file for setting up Cal/Diags
 15 datalogging, a second datalog setup file for setting up datalogging for functional tests, and a third datalog setup file for setting up datalogging for parametric tests.

Datalog Format Block

[0049] In one embodiment, a datalog format block is used to define a datalog format. The definitions of the format may include a format name, a format DLL name, a message
 20 format string and optional user-defined format parameters. If the format DLL name is GeneralFormat, the predefined GeneralFormat is used; otherwise, a user-defined datalog format is used. In the latter case, the user provides a datalog format DLL in the system test directories. The following Datalog API is called to define the datalog format block:

```

    void DatalogManager::addFormat(const OFCString &formatName,
25    const OFCString &typeName,
    const OFCString &format,
    const DatalogProperties_t properties);

```

and

```

    Format SignalInfoFormat
30    {
        DLL "GeneralFormat";
        Format "$SignalInfo";
    }

```

defines a format `SignalInfoFormat`, which makes use of the predefined `GeneralFormat`. Its format string is `$SignalInfo`, in which `$SignalInfo` is a token which may be replaced by the datalog variable `SignalInfo` in the passed-in datalog record (`DatalogValues`) during runtime. A datalog format block can be used to remove an existing datalog format from the datalog system. In this scenario, the following datalog API is called.

```
void DatalogManager::removeFormat(const OFCString &formatName);
```

and

```
Format AnExistingFormat Disabled;
```

removes datalog format *AnExistingFormat* from the datalog system.

Datalog Stream Block

[0050] In another embodiment, a datalog stream block is used to define a datalog stream. The definition of the datalog stream includes a stream name, a stream DLL name and optional user-defined stream parameters. If the stream DLL name is `FileStream` or `ConsoleStream`, a built-in datalog stream `FileStream` or `ConsoleStream` is used; otherwise, a user-defined datalog stream is used. In the latter case, the user provides a datalog stream DLL in the system test directories. The following Datalog API is called to define the datalog stream block:

```
void DatalogManager::addStream(const OFCString &streamName,  
                                const OFCString &typeName,  
                                const DatalogProperties_t properties);
```

and

```
Stream FunctionalTestStream  
{  
    DLL "FileStream";  
    FileName "datalog<DutID>.log";  
    Overwrite "0";  
}
```

defines a stream `FunctionalTestStream`, which makes use of the built-in stream `FileStream` with optional parameter `FileName` and `Overwrite`. Note that `FileStream` supports a file

name with automatic variables <DutID> and <TimeStamp>. The <DutID> is replaced by the current DUT ID during runtime while the <TimeStamp> is replaced by the current date and time with format such as Thu_Nov_13_06_40_28_2003.

[0051] A datalog stream block can be used to remove an existing datalog stream from the datalog system. In this case, the following datalog API is called:

```
void DatalogManager::removeStream(const OFCString &streamName);
```

and

```
10 Stream AnExistingStream Disabled;
```

removes datalog stream *AnExistingStream* from the datalog system.

Datalog Type Block

[0052] In a different embodiment, a datalog type block is used to define user-specific properties for a datalog type. The following Datalog API is called to define the datalog type block:

```
IDatalogType *DatalogManager::getType(const OFCString &typeName)
const;
```

```
20 void IdatalogType::addProperty(const OFCString &propName,
                                const OFCString &propValue);
```

and

```
25 Type com.Advantest.oai.TestClasses.FunctionalTest
{
    Mode "Detailed";
}
```

defines a property *Mode* with value "*Detailed*" for datalog type
30 "*com.Advantest.oai.TestClasses. FunctionalTest.*"

Datalog Format Map Block

[0053] In yet another embodiment, a datalog format map block is used to define a datalog format map. The definition of the Format Map includes a format map name, a datalog type name and its event name to be mapped, and a mapped format name. The
35 following Datalog API is called to define the datalog format map block:

15

```

void DatalogManager::createFormMap(const OFCString &mapName,
                                   const OFCString &typeName,
                                   const OFCString &eventName,
                                   const OFCString &formatName);
5
and
    FormatMap SignalFormatMap
    {
        Type com.Advantest.oai.TestClasses.FunctionalTest;
10        Event DumpSignal;
        Format SignalInfoFormat;
    }

```

defines a format map *SignalFormatMap*, which maps datalog type
15 *com.Advantest.oai.TestClasses.FunctionalTest* and its event *DumpSignal* to format
SignalInfoFormat.

[0054] A datalog format map block may be used to remove an existing datalog format
map from the datalog system. In this case, the following datalog API

```

20 void DatalogManager::removeFormatMap(const OFCString &mapName);

```

is called to perform the task.

Datalog Format Group Block

[0055] In one embodiment, a datalog format block is used to define a datalog format
25 group. The definition of the *Format Group* includes a format group name, a list of names
of format maps that the format group contains, and a list of names of streams to which
messages generated from the format group are exported. The following Datalog APIs are
called to define the datalog format group block:

```

30 void DatalogManager::createFormatGroup(const OFCString &groupName,
    const OFCStringVec_t &mapNames);
void DatalogManager::setStream(const OFCString &groupName,
    const OFCString &streamName);

```

The first call creates a format group that contains a vector of formats, and the second call
35 adds a datalog stream into the created format group. For example,

```

FormatGroup FunctionalTestGroup
{
    FormatMap SignalFormatMap;

```

```

        FormatMap ResultFormatMap;
        Stream FunctionalTestStream;
        Stream ConsoleStream;
    }

```

5 defines a format group FunctionalTestGroup, which includes a format map SignalFormatMap and a ResultFormatMap. The messages generated from the format group are exported to datalog stream FunctionalTestStream and ConsoleStream.

[0056] A datalog format map block may be used to remove an existing datalog format group from the datalog system. In this case, the datalog API

10

```

void DatalogManager::removeFormatGroup(const OFCString
&groupName);

```

is called to perform the function.

15 Datalog System Application Examples

[0057] In the following examples, proxy objects, such as DatalogManagerProxy, DatalogHandlerProxy, and DatalogFilterProxy, are used by applications running the system controller to remotely control the current state and operation of the datalog framework on one or more site controllers. These objects act as remote proxies for the real objects on the site controllers and provide a transparent communication channel to these objects that allows applications on the system controller to deal with local object rather than the complexities of communication protocols.

20

[0058] Figure 6 illustrates an implementation of enabling datalogging dynamically according to an embodiment of the present invention. The group of collaborating objects for achieving this task includes a Datalog graphical user interface (DatalogGUI) object 602, a DataloggerProxy object 604 (also referred to as DatalogManagerProxy), a test object 606 and a Datalogger object 608 (also referred to as DatalogManager). The sequence diagram illustrates the steps for enabling a datalog system dynamically. As shown in Figure 6, each step of the sequence diagram is described as follows:

25

- 30 1. In step 1, the DatalogGUI 602 or other tester GUI (e.g., Test Control Panel) includes a button or menu item Enable Datalog. When a user clicks on the button or menu item, the DatalogGUI 602 searches the DataloggerProxy object 604 from a test plan server proxy (TPSPProxy).

2. The DatalogGUI 602 invokes a `setEnabled()` method on the returned DataloggerProxy 604, passing in parameter value 'true', which means enabling the datalog system.
3. With the proxy model, the `setEnabled()` method of the DataloggerProxy 604
5 invokes the `setEnabledTest()` method of the Datalogger 608 to enable the datalog system.
4. The test object 606 calls an `enteringTest()` method to send out a datalog event to the Datalogger 608.
5. The Datalogger 608 checks whether it is enabled by calling an `isEnabled()` method.
- 10 6. If the Datalogger 608 is enabled, it invokes a `log()` method to start datalogging.

[0059] Figure 7 illustrates an implementation of disabling datalogging dynamically according to an embodiment of the present invention. The group of collaborating objects for achieving this task includes a DatalogGUI object 702, a DataloggerProxy object 704, a test object 704 and a Datalogger object 708. The Datalogger 708 may be used to disable
15 all datalogging. The sequence diagram illustrates the steps for disabling a datalog system dynamically. As shown in Figure 7, each step of the sequence diagram is described as follows:

1. In step 1, the DatalogGUI 702 or other tester GUI (e.g., Test Control Panel) includes a button or menu item Disable Datalog. When a user clicks on the button or
20 menu item, DatalogGUI searches the singleton DataloggerProxy object 704 from a TPSPProxy.
2. The DatalogGUI 702 invokes a `setEnabled()` method on the returned DataloggerProxy 704, passing in parameter value 'false', which means disabling the datalog system.
- 25 3. With the proxy model, the `setEnabled()` method of the DataloggerProxy 704 invokes the `setEnabledTest()` method of the Datalogger 708 to disable the datalog system.
4. The test object 706 calls an `enteringTest()` method to send out a datalog event to the Datalogger 708.
- 30 5. The Datalogger checks whether it is enabled. Since it is disabled, no datalogging may occur.

[0060] Figure 8 illustrates an implementation of modifying datalog format dynamically according to an embodiment of the present invention. The group of collaborating objects

for achieving this task includes a DatalogGUI object 802, a DatalogFormatterProxy object 804, a DatalogHandler object 806, and a DatalogFormatter object 808. The sequence diagram illustrates the steps for changing the datalog format dynamically. The DatalogHandler object 806 allows users to combine datalog stream and format specifications together in a single entitle that the DatalogManager may use to pass datalog events to. As shown in Figure 8, each step of the sequence diagram is described as follows:

1. In step 1, the DatalogGUI 802 displays a list of available datalog formatters associated with a selected datalog handler. A user may select any datalog formatter by opening a pull-down menu, and selects the menu item Edit. The implementation of DatalogGUI associates each datalog formatter item in the list with the corresponding datalog formatter proxy reference (by using user data parameter). Next, the DatalogFormatterProxy object 804 is obtained.
2. The DatalogGUI 802 invokes a getFormat() method using the DatalogFormatterProxy object 804 to get the format string and related arguments.
3. With the proxy model, the getFormat() method of the DatalogFormatterProxy 804 invokes the corresponding getFormat() method of the DatalogFormatter 808.
4. The DatalogGUI 802 displays the format and argument of the datalog event with the selected formatter. The user may change the format and argument according to a set of predetermined datalog format requirements.
5. When the user applies the modified formatter into the datalog system, the DatalogGUI 802 invokes a setFormat() method of the DatalogFormatterProxy 804 and passes in the modified format string and arguments.
6. With the proxy model, DatalogFormatterProxy.setFormat() method of the DatalogFormatterProxy 804 invokes the corresponding setFormat() method of the DatalogFormatter 808.
7. Next, when the DatalogHandler 806 invokes a getOuptut() method on the modified datalog formatter to get formatted message with the datalog event, the modified formatter is applied.

[0061] Figure 9 illustrates an implementation of assigning datalog output streams dynamically according to an embodiment of the present invention. The group of collaborating objects for achieving this task includes a DatalogGUI object 902, a DatalogManagerProxy object 904, a IDatalogHandlerProxy object 906, a IDatalogHandler

object 908, and a DatalogManager object 910. The sequence diagram illustrates the steps for assigning datalog output streams to associate with a datalog handler dynamically. As shown in Figure 9, each step of the sequence diagram is described as follows:

1. In step 1, when a user selects a datalog handler to edit its associated datalog
5 output stream, the DatalogGUI 902 displays a panel, which comprises two lists: a first list displays datalog stream names in the datalog system, and a second list displays the datalog stream names associated with the datalog handler. In addition, the DatalogGUI also includes Add, Remove, Apply, OK and Cancel buttons.
2. The DatalogGUI 902 calls a `getStreamNames()` method of the
10 DatalogManagerProxy 904.
3. With the proxy model, the `getStreamNames()` method of the DatalogManagerProxy 904 invokes the corresponding `getStreamNames()` method of the DatalogManager 910 to get all datalog stream names registered in the datalog system.
4. The DatalogGUI 902 displays the retrieved datalog stream names in the first list.
15
5. The DatalogGUI 902 then calls a `getStreamNames()` method of the IDatalogHandlerProxy 906.
6. With the proxy model, the corresponding `getStreamNames()` method of the IDatalogHandlerProxy 906 invokes the corresponding `getStreamNames()` method of
20 the IDatalogHandler 908 to get all datalog stream names currently associated with the selected datalog handler.
7. The DatalogGUI 902 displays the retrieved datalog stream names associated with the selected datalog handler in the second list.
8. Then, the user edits the associated datalog stream list using the buttons Add and
25 Remove. After the user finishes editing and clicks an Apply or Ok button to apply the modified stream to the datalog system, the DatalogGUI 902 invokes a `setStreams()` method on the IDatalogHandlerProxy 906 and passes to it the modified datalog stream names.
9. With the proxy model, the `setStreams()` method of the IDatalogHandlerProxy 906
30 invokes the corresponding `setStreams()` method of the IDatalogHandler 908.
10. Then, the IDatalogHandler 908 gets the associated datalog stream object reference for each datalog stream name from the DatalogManager 910.

11. The IDatalogHandler 908 then adds the stream into the stream vector. Afterwards, the IDatalogHandler 908 sends out newly selected datalog streams in the stream vector for its datalog output.

- [0062] Figure 10 illustrates an implementation of disabling datalog output stream dynamically according to an embodiment of the present invention. The group of collaborating objects for achieving this task includes a DatalogGUI object 1002, a IDatalogStreamProxy object 1004, a IDatalogHandler object 1006 and a IDatalogStream object 1008. The sequence diagram illustrates the steps for disabling a datalog output stream dynamically. As shown in Figure 10, each step of the sequence diagram is described as follows:
1. In step 1, the DatalogGUI 1002 displays a list of available datalog streams. A user may select any datalog stream by opening a pull-down menu, and selects a menu item Disable. The implementation of the DatalogGUI 1002 associates each datalog stream item in the list with the corresponding IDatalogStreamProxy reference 1004 (by using user data parameter). Then, the IDatalogStreamProxy 1004 is obtained.
 2. The DatalogGUI 1002 invokes a disable() method on the IDatalogStreamProxy object 1004 obtained.
 3. The disable() method of the IDatalogStreamProxy 1004 invokes the corresponding disable() method of the IDatalogStream 1008 through the proxy model.
 4. The IDatalogHandler 1006 calls a write() method to write out a formatted message to the stream.
 5. The IDatalogStream 1008 checks whether it is enabled. If it is enabled, the output stream may be modified.

- [0063] Figure 11 illustrates an implementation of adapting new source dynamically according to an embodiment of the present invention. The group of collaborating objects for achieving this task includes a DatalogGUI object 1102, a DatalogManagerProxy object 1104, a test object 1106, a Datalogger object 1108, a DatalogManager object 1110, a DatalogHandlerProxy object 1112, a DatalogHandler object 1114, a DatalogFilterProxy object 1116, a DatalogFilter object 1118, and a DatalogFormatter object 1120. The sequence diagram illustrates the steps for adapting new source dynamically. The DatalogFilter object 1118 allows users to selectively filter datalog events by type and source identifiers. This filtering allows users to turn on and off specific datalog events. As shown in Figure 11, each step of the sequence diagram is described as follows:

1. In step 1, the DatalogGUI 1102 finds the DatalogManagerProxy object 1104, and calls its addHandler() method to register a new DatalogHandler instance for the new source. Note that in this approach, the DatalogHandler 1114 associates an event with a different Datalog Formatter. With this model, a new DatalogHandler instance is
5 created to work with the new source (test class). The newly created DatalogHandlerProxy instance is returned.
2. With the proxy model, the addHandler() method of the DatalogManagerProxy 1104 invokes the corresponding addHandler() method of the DatalogManager 1110.
3. Next, the addHandler() method of the DatalogManager 1110 creates a new
10 instance of DatalogHandler, then it registers the newly created DatalogHandler instance to the DatalogManager 1110.
4. Then, the DatalogGUI 1102 associates the datalog filter proxy with the newly created datalog handler through the DatalogHandlerProxy 1104 returned in step 1.
5. With the proxy model, the getFilter() method of the DatalogHandlerProxy 1112
15 invokes the corresponding getFilter() method of the DatalogHandler 1114.
6. With the DatalogFilterProxy 1116, the DatalogGUI 1102 calls its enableTest() method.
7. With the proxy model, the enableTest() method of the DatalogFilterProxy 1116
20 invokes the corresponding enableTest() method of the DatalogFilter 1118. As a result, the newly created datalog handler instance handles a selected event of the new test source.
8. The DatalogGUI 1102 sets the formats for each selected datalog event of the newly created datalog handler through its proxy. For this new test source, the format of a new argument may be specified as Test.FooArgument. In the new test source, an
25 interface IProperty may be implemented, which is shown as follows:

```

class IProperty
{
public:
30         OFCString &getProperty(const OFCString &name) const;
        void setProperty(const OFCString &name, const OFCString
        *value);
        OFCStringVec_t getPropertyNames() const;
        };
35     where FooArgument is one of the properties in the new test source.

```

9. The DatalogHandlerProxy 1112 calls a setFormat() method through the corresponding DatalogHandler 1114.

10. The DatalogHandler 1114 creates a DatalogFormatter for the selected event and adds it into the event route map. Next, the newly created datalog handler processes the datalog event from the new source. Then, the test plan with the new source test may start.

11. Once the TPS reaches the end of execution of the new source test, it sends an exitingTest() datalog request to the datalog system.

12. The Datalogger 1108 accepts the request, and transfers it to the datalog event to be logged.

13. The Datalogger 1108 forwards the new datalog event to the DatalogManager 1110 to dispatch.

14. The DatalogManager 1110 publishes the event to the registered datalog handlers one by one. Once a datalog handler is able to process the event, the publishing process may be stopped.

15. When the event is published to each datalog handler, the handler checks whether the event is loggable. The event is loggable if the newly created DatalogHandler instance is able to service the new test source.

16. The DatalogHandler 1114 finds the DatalogFormatter 1120 associated with the current event, and calls its getOutput() method to get the formatted message based on the defined format. The getOutput() method fetches the new parameter Test.FooArgument value by calling

```
event.getDatalogger->getTest()->getProperty("FooArgument");
```

the formatted output message is then forwarded to the output datalog stream.

[0064] There are number of benefits achieved by the disclosed datalog framework.

First, the datalog framework is independent of the sources, nature and contents of the datalog events. This allows new datalog sources and events to be added without modifying the datalog framework. In addition, the formatting of datalog output is independent of the datalog framework. While the GeneralFormat is provided with the system, it is specified as a dynamically linked library (DLL) parameter to the Format block, and it is not "hard-coded" into the framework. Moreover, the formatting of datalog events is configurable by

the end user. For example, events may be formatted into human readable text, comma-separated values for spreadsheets or databases, application-specific text or binary formats for further processing, or they may even be ignored entirely. The datalog framework is independent of the destination of datalog streams. Furthermore, formats and output
5 streams are extendable. That is, users may add new IDatalogFormat and IDatalogStream implementations to the system without modifying the framework. Test classes (and other datalog sources) and their corresponding datalog events are also extendable so the framework does not need to know about specific source or event types. As users add new Test classes with new datalog event types, no modification of the datalog framework is
10 required.

[0065] One skilled in the relevant art will recognize that many possible modifications of the disclosed embodiments may be used, while still employing the same basic underlying mechanisms and methodologies. The foregoing description, for purpose of explanation, has been written with references to specific embodiments. However, the
15 illustrative discussions above are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described to explain the principles of the invention and its practical applications, and to enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to
20 the particular use contemplated.

CLAIMS

What is claimed is:

- 5 1. A method for communicating test information from a source to a destination, comprising:
- providing a modular test system, wherein the modular test system comprises a system controller for controlling at least one site controller, the at least one site controller for controlling at least one test module;
- 10 providing a datalog framework for supporting extension of user-defined datalog formats;
- providing support classes for supporting user-initiated datalog events;
- receiving a datalog event requesting for communicating input test information from the source to the destination;
- 15 configuring output test information based upon the destination, the datalog framework and the support classes; and
- transferring the output test information to the destination.
2. The method of claim 1, wherein the datalog framework comprises:
- 20 a source interface for supporting the input test information of individual sources from different vendors;
- a type interface for representing classes of datalog sources and destinations;
- a format interface for providing formatting capabilities;
- a stream interface for transferring formatted test information to different
- 25 destinations; and
- a datalog manager for managing datalog events in response to the corresponding sources, streams, formats and destinations.
3. The method of claim 1, wherein the support classes comprise:
- 30 a datalog value class for storing header, type and event information;
- a datalog format map class for combining a type and a name of a user event;
- a datalog format group class for bundling a group of format maps;
- a user format class for supporting user-defined format classes; and

a general format class for supporting a set of predefined format classes.

4. The method of claim 1, wherein the step of providing the datalog framework further comprises:
 - 5 creating one or more datalog streams;
 - creating one or more user formats;
 - creating one or more datalog format maps;
 - creating one or more datalog format groups; and
 - linking the one or more datalog streams, user formats, datalog format maps and
 - 10 datalog format groups to a corresponding datalog manager.
5. The method of claim 1, wherein the datalog framework is independent of the source and contents of input test information.
- 15 6. The method of claim 1, wherein the datalog framework is independent of the destination and contents of output test information.
7. The method of claim 1, wherein formats of the output test information are configurable by users.
- 20 8. The method of claim 1, wherein the step of configuring comprises:
 - creating a datalog value object, the datalog value object including type and event information;
 - determining a datalog format object based upon the corresponding destination,
 - 25 datalog framework and support classes; and
 - formatting the output test information using the datalog format object.
9. A modular test system, comprising:
 - a system controller;
 - 30 at least one site controller coupled to the system controller;
 - at least one test module and its corresponding device under test (DUT);
 - a datalog framework configured to support extension of user-defined datalog formats;

one or more support classes configured to support user-initiated datalog events;
means for receiving a datalog event requesting for communicating input test
information from the source to the destination;

means for configuring output test information based upon the destination, the
5 datalog framework and the support classes; and
means for transferring the output test information to the destination.

10. The system of claim 9, wherein the datalog framework comprises:
a source interface for supporting the input test information of individual sources
10 from different vendors;
a type interface for representing classes of datalog sources and destinations;
a format interface for providing formatting capabilities;
a stream interface for transferring formatted test information to different
destinations; and
15 a datalog manager for managing datalog events in response to the corresponding
sources, streams, formats and destinations.

11. The system of claim 9, wherein the support classes comprise:
a datalog values class for storing header, type and event information;
20 a datalog format map class for combining a type and a name of a user event;
a datalog format group class for bundling a group of format maps;
a user format class for supporting user-defined format classes; and
a general format class for supporting a set of predefined format classes.

25 12. The system of claim 9, wherein providing the datalog framework further
comprises:
means for creating one or more datalog streams;
means for creating one or more user formats;
means for creating one or more datalog format maps;
30 means for creating one or more datalog format groups; and
means for linking the one or more datalog streams, user formats, datalog format
maps and datalog format groups to a corresponding datalog manager.

13. The system of claim 9, wherein the datalog framework is independent of the source and contents of input test information.
14. The system of claim 9, wherein the datalog framework is independent of the destination and contents of output test information.
15. The system of claim 9, wherein formats of the output test information are configurable by users.
16. The system of claim 9, wherein the means for configuring output test information comprise:
- means for creating a datalog value object, the datalog value object including type and event information;
 - means for determining a datalog format object based upon the corresponding destination, datalog framework and support classes; and
 - means for formatting the output test information using the datalog format object.

1/11

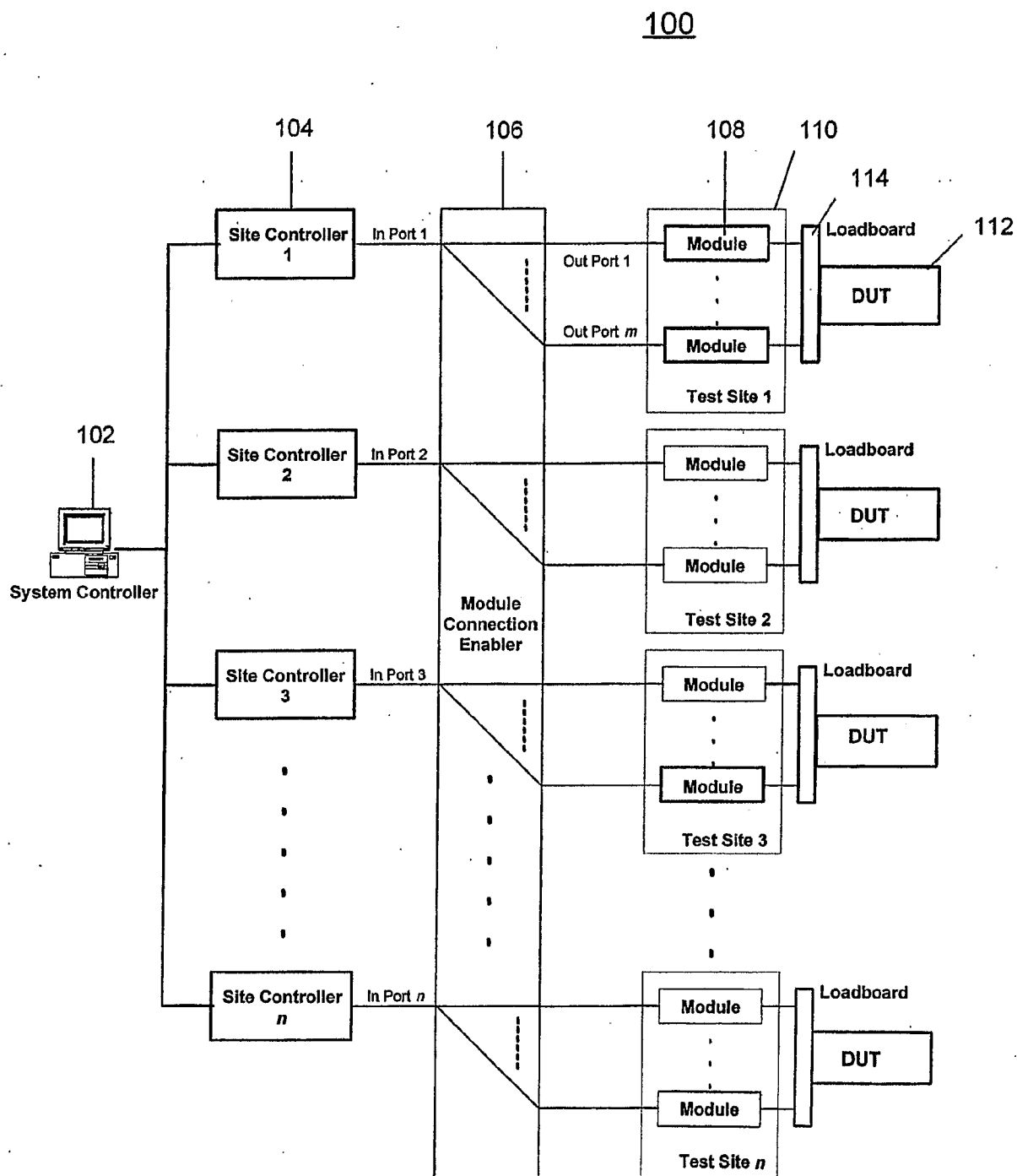


Figure 1

2/11

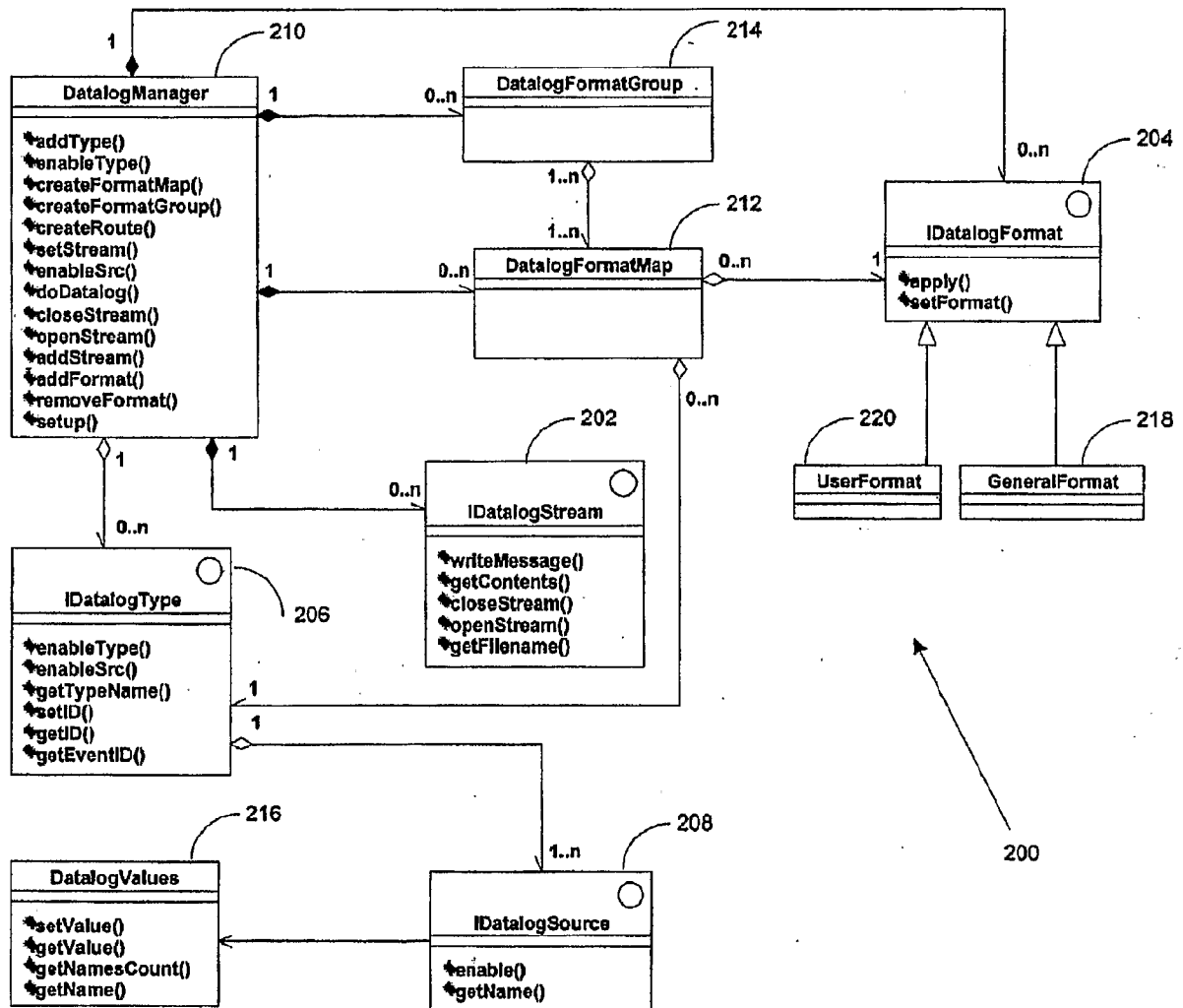


Figure 2

3/11

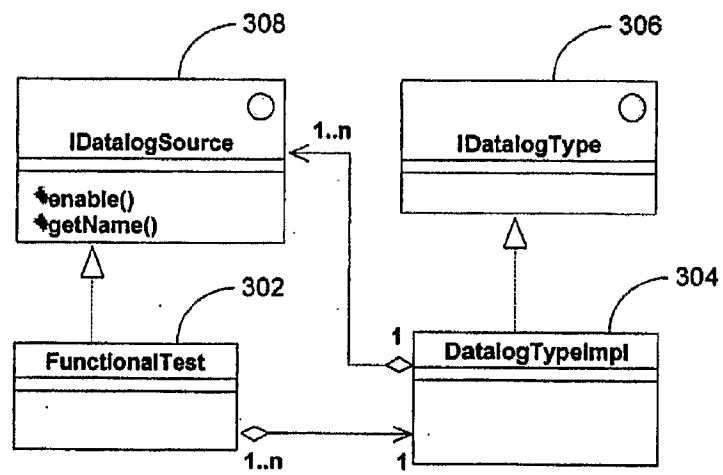


Figure 3

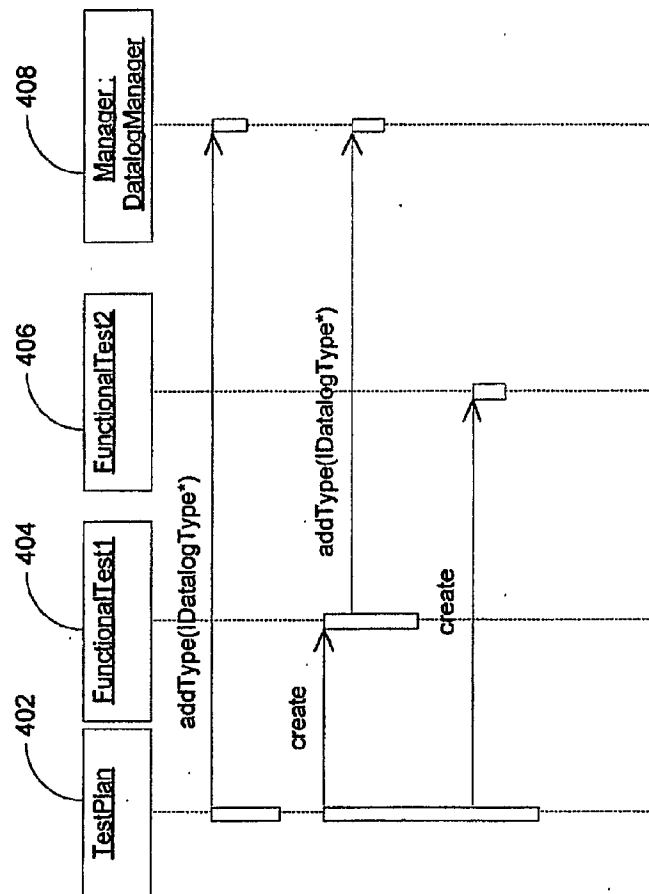


Figure 4

5/11

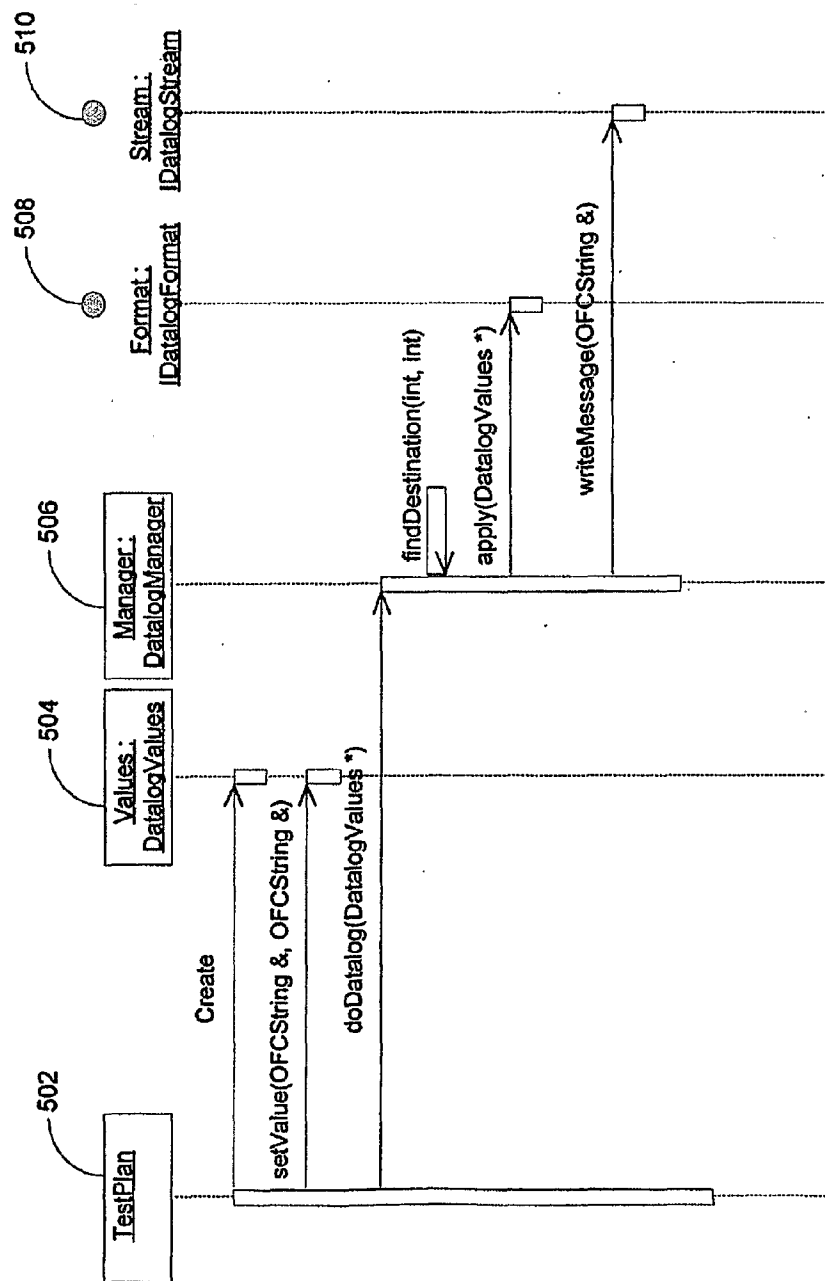


Figure 5

6/11

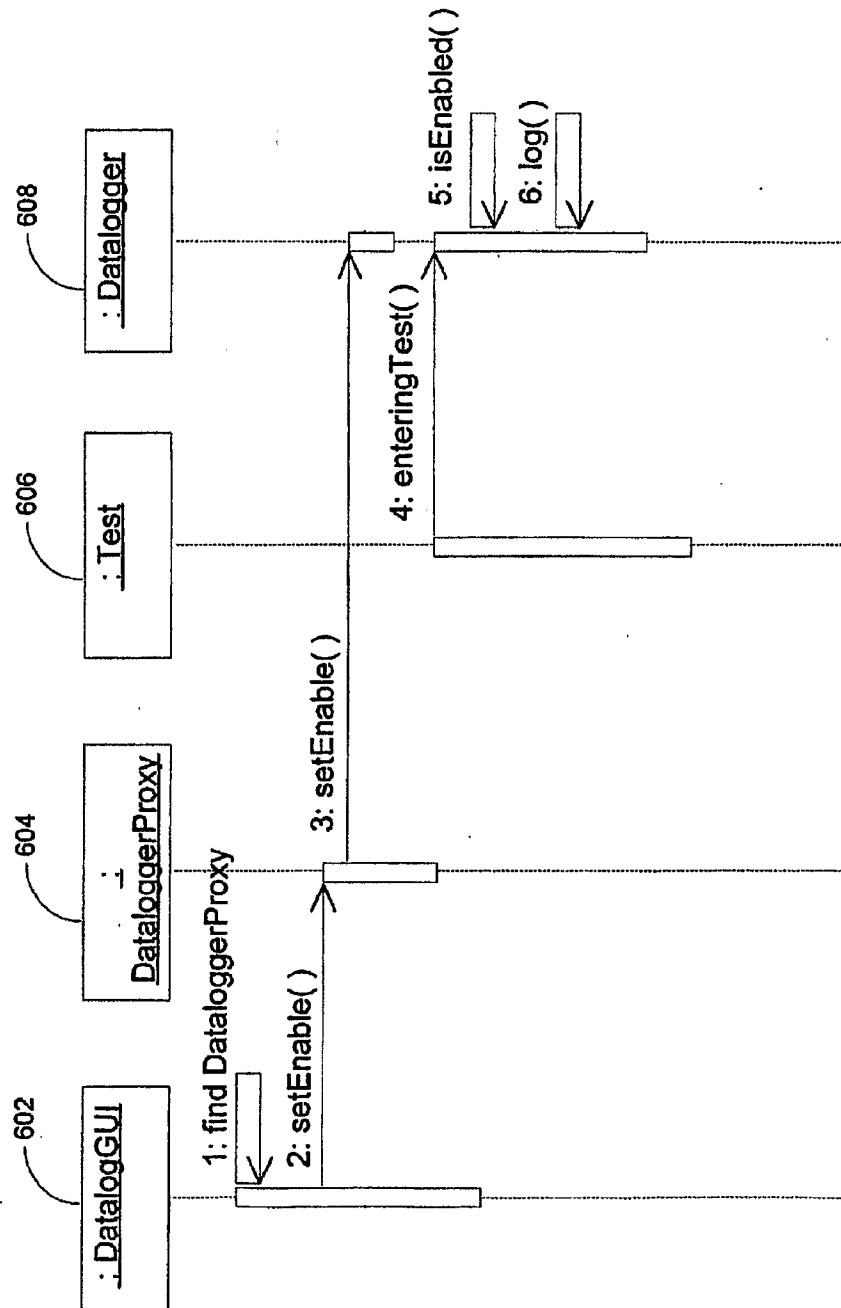


Figure 6

7/11

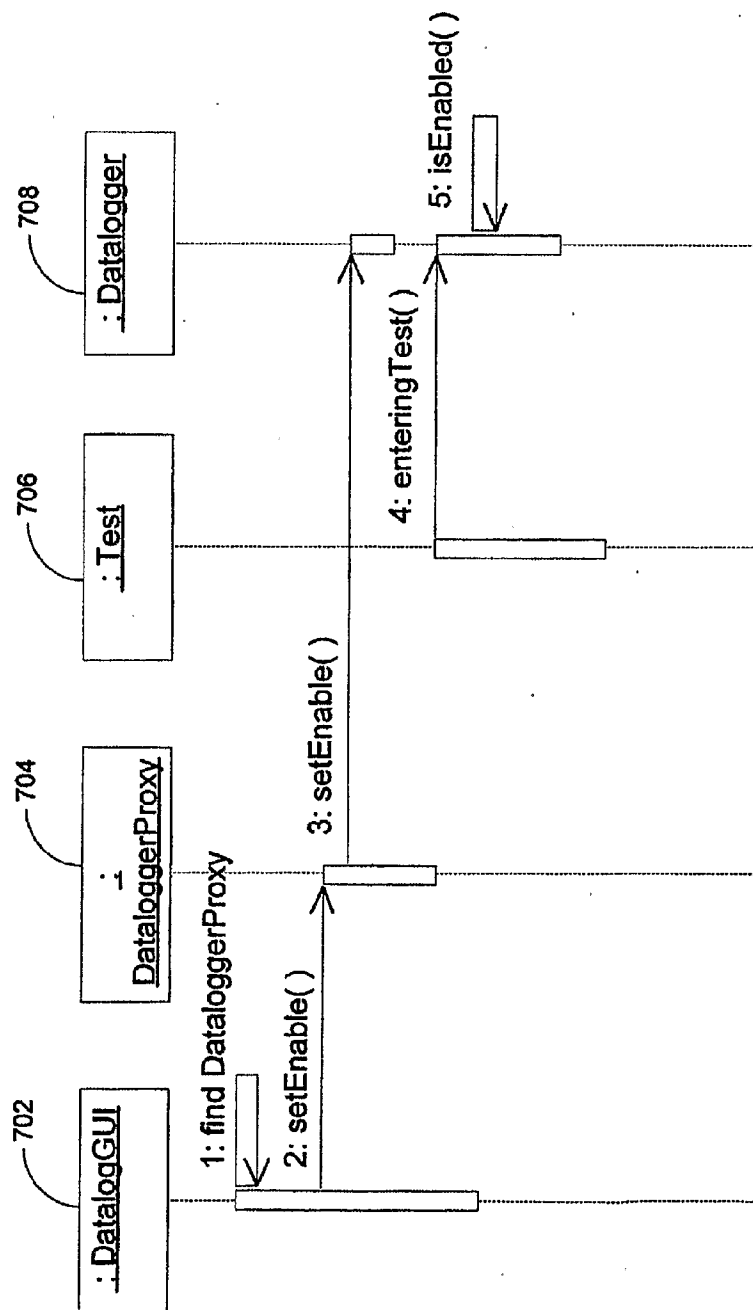


Figure 7

8/11

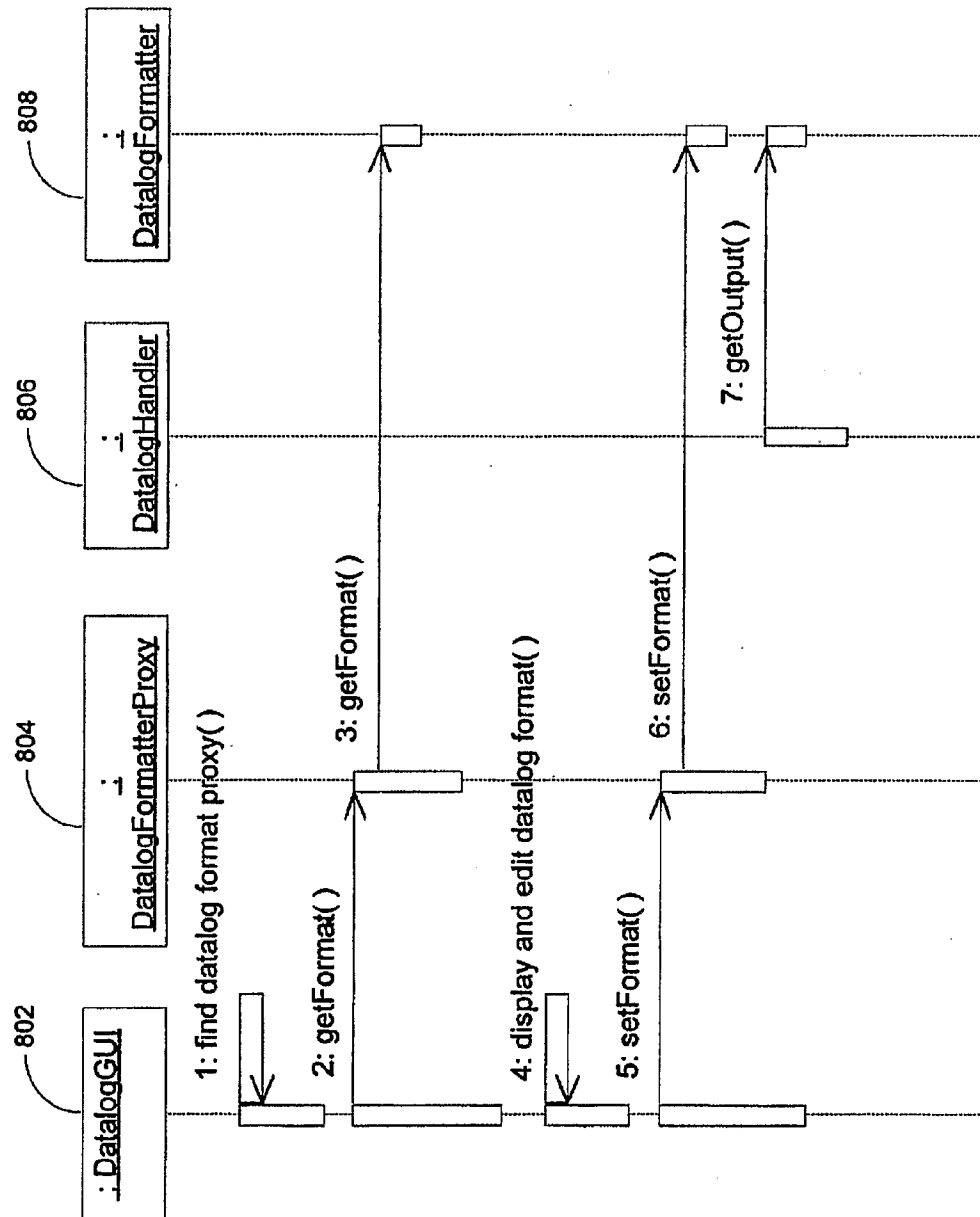


Figure 8

9/11

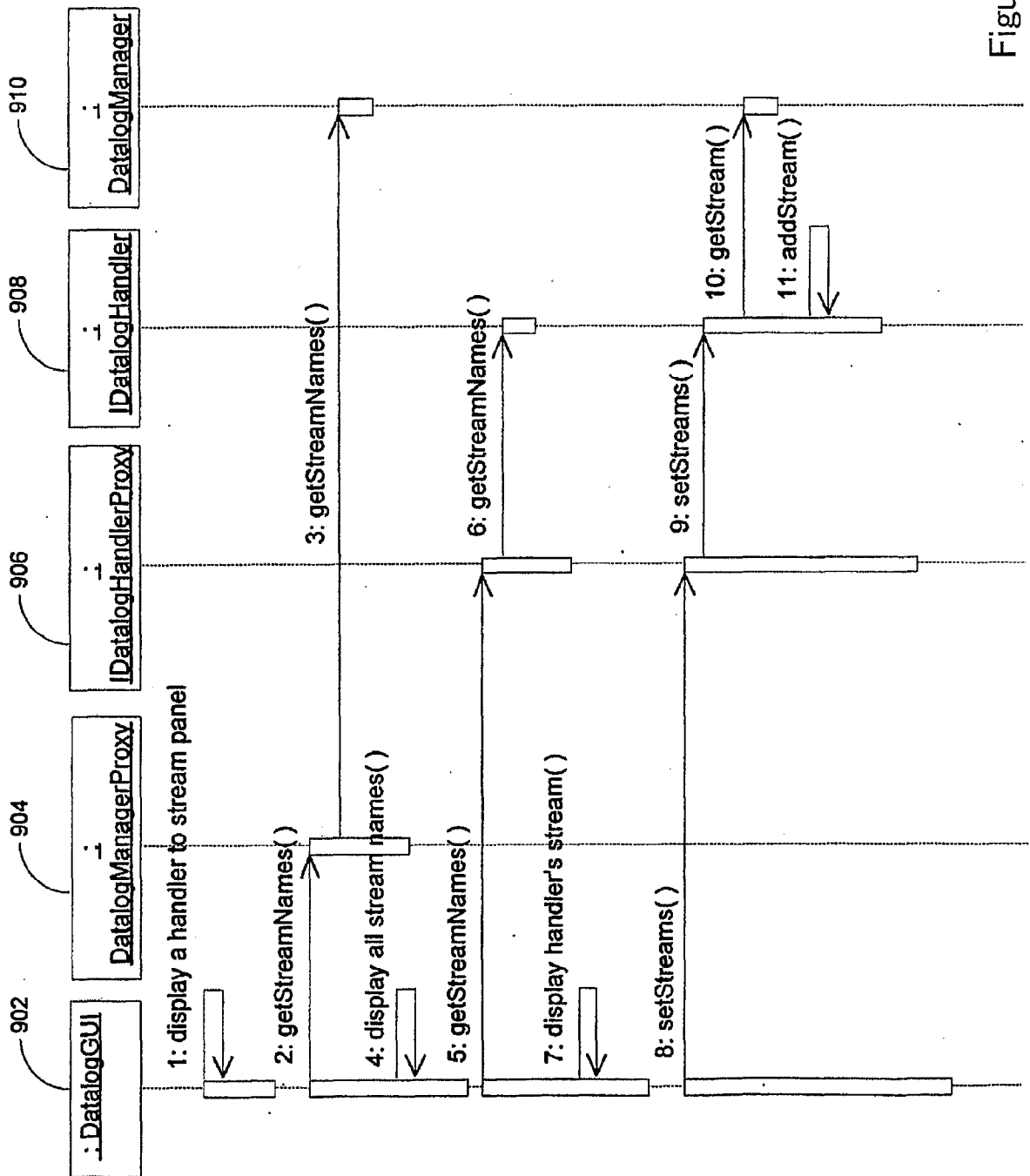


Figure 9

10/11

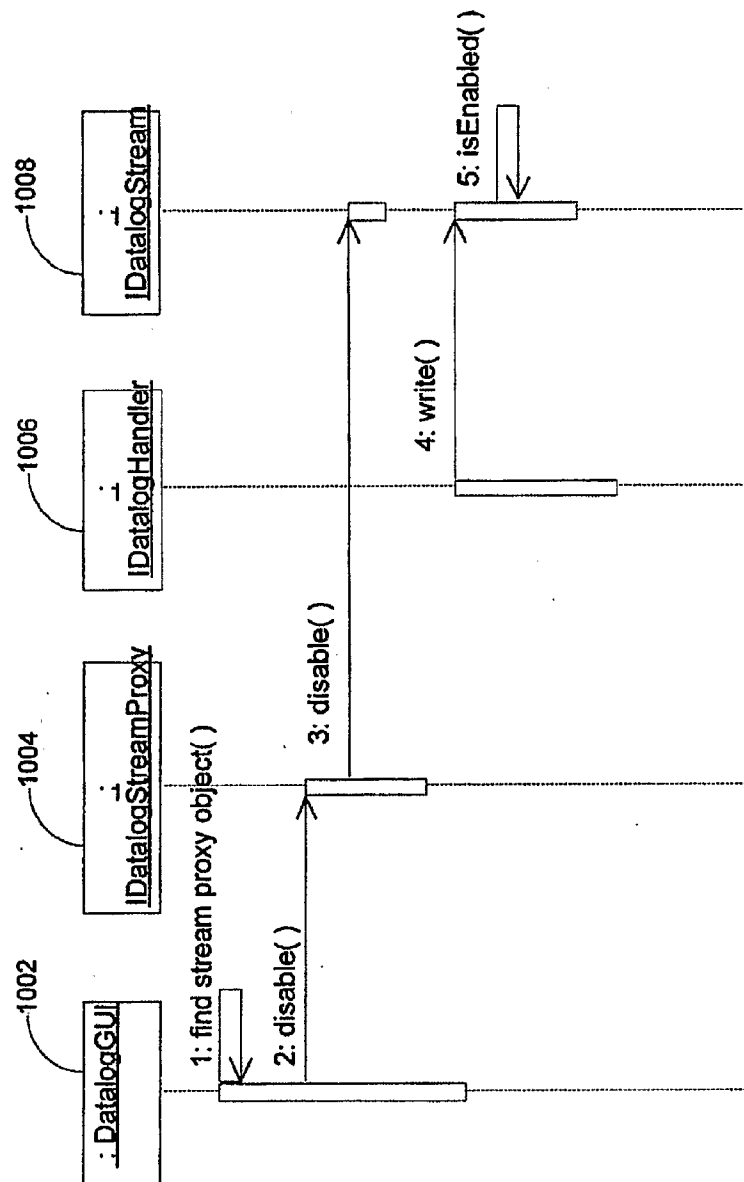


Figure 10

11/11

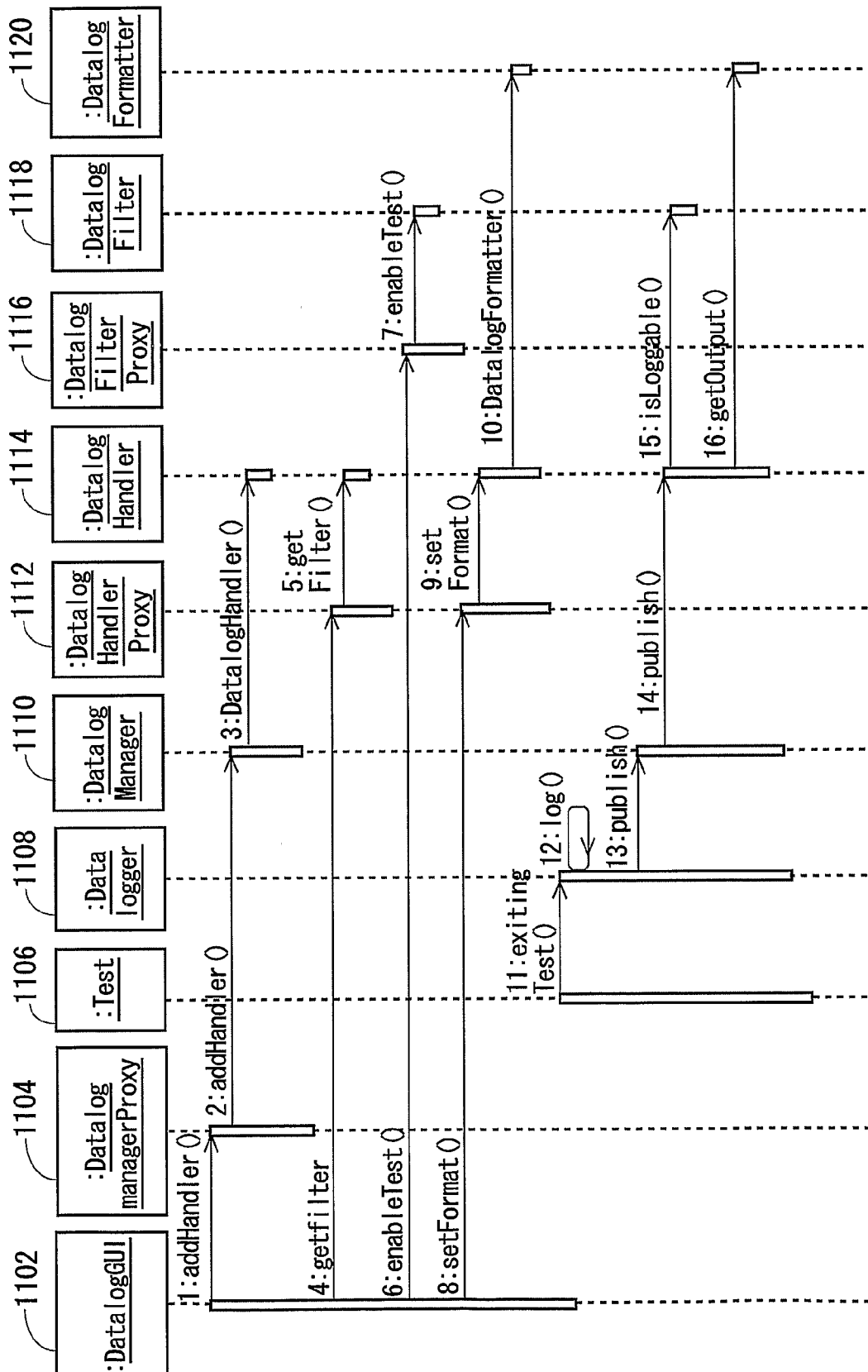


Figure 11

INTERNATIONAL SEARCH REPORT

International Application No

PCT/JP2005/009810

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G01R31/319

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 G01R

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6 028 439 A (ARKIN ET AL) 22 February 2000 (2000-02-22) column 2, line 65 - column 4, line 17 column 4, line 44 - column 9, line 38 abstract; claims 1-13; figures 1-9 column 14, line 34 - column 16, line 12	1-16
X	US 2003/005375 A1 (KRECH ALAN S ET AL) 2 January 2003 (2003-01-02) abstract; claims 1-9; figures 1-6 paragraphs '0020!, '0027! - '0034! paragraph '0079! ----- --/--	1-16

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search

29 July 2005

Date of mailing of the international search report

08/08/2005

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Nadal, R

INTERNATIONAL SEARCH REPORT

International Application No
PCT/JP2005/009810

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,X	WO 2004/072670 A (ADVANTEST CORPORATION; KRISHNASWAMY, RAMACHANDRAN; SINGH, HARSANJEET;) 26 August 2004 (2004-08-26) abstract; claims 1-23; figures 1-9 page 10, lines 17-23 page 11, lines 17-30 page 12, line 12 - page 13, line 17 page 103, line 6 - page 104, line 6 page 117, line 7 - page 125, line 6 page 145, line 6 - page 146, line 4 page 161, line 20 - page 163, line 31 page 205, line 27 - page 211, line 5 -----	1-16
P,X	WO 2004/072669 A (ADVANTEST CORPORATION; PRAMANICK, ANKAN; ELSTON, MARK; CHEN, LEON; SAU) 26 August 2004 (2004-08-26) abstract; claims 1-24; figures 1-8 page 3, line 5 - page 6, line 11 page 11, line 5 - page 13, line 15 page 21, line 6 - page 30, line 5 -----	1-16

INTERNATIONAL SEARCH REPORT

International Application No
PCT/JP2005/009810

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 6028439	A	22-02-2000	EP 1032844 A1 JP 2001522049 T WO 9923499 A1	06-09-2000 13-11-2001 14-05-1999
US 2003005375	A1	02-01-2003	NONE	
WO 2004072670	A	26-08-2004	US 2004193990 A1 US 2004210798 A1 WO 2004072669 A1 WO 2004072670 A1 US 2004225465 A1 US 2004225459 A1 US 2005039079 A1 US 2005022087 A1 US 2005154550 A1 US 2005154551 A1 WO 2004088339 A1 WO 2004090562 A1 US 2004255216 A1	30-09-2004 21-10-2004 26-08-2004 26-08-2004 11-11-2004 11-11-2004 17-02-2005 27-01-2005 14-07-2005 14-07-2005 14-10-2004 21-10-2004 16-12-2004
WO 2004072669	A	26-08-2004	US 2004193990 A1 US 2004210798 A1 WO 2004072669 A1 WO 2004072670 A1 US 2004225465 A1 US 2004225459 A1 US 2005039079 A1 US 2005022087 A1 US 2005154550 A1 US 2005154551 A1 WO 2004088339 A1 WO 2004090562 A1 US 2004255216 A1	30-09-2004 21-10-2004 26-08-2004 26-08-2004 11-11-2004 11-11-2004 17-02-2005 27-01-2005 14-07-2005 14-07-2005 14-10-2004 21-10-2004 16-12-2004